Experiment 1: A basic Web Server

1 Introduction

In this lab you will implement a *server* and a *client* using sockets in Python. Note: Python 2 is deprecated so we accept only Python 3 code.

At the end of this lab you should know how to:

1. Explain how a basic web server works;
2. Design the client and the server; and
3. Implement and test the network application using sockets.

2 Background

2.1 Networking basics

If you have taken prior courses in Telecommunications such as ECSE 316 there is a good chance you do not need to read additional material to do the assignment. If you have not taken a related course or want a brief review of the fundamentals of what a server and a client do and an HTTP requests format guide you can read the relative sections from Chapters 1 & 2 of Kurose and Ross.

2.2 Socket programming

Sockets are the programming mechanism used to implement network applications. If you have previously written an application that reads or writes from a file on disk, then you know you first open a file handle, then read/write using the file handle, and finally close the file handle when you are done. Sockets work in a similar way, but instead of reading/writing to a file on disk, they allow you to read/write to another process (usually running on a different machine).

For more on socket programming read material available at:
https://docs.python.org/2/howto/sockets.html

2.3 Summary of background and pre-requisites

Before proceeding with the rest of this lab document make sure you have accomplished these first steps:

1. You are familiar with basic Python programming, including how to compile and execute a program from the command line. We also recommend that you are familiar with an *integrated development environment (IDE)*, such as PyCharm, for debugging.
2. You have read the background on the server-client basics, HTTP requests and Python socket programming mentioned above.

3 Lab requirements

Write a program using Python sockets that:

(i) creates a connection socket when contacted by a client
(ii) receives the HTTP request from this connection
(iii) parses the request to determine the specific file being requested
(iv) gets the requested file from the server's file system
(v) creates an HTTP response message consisting of the requested file preceded by header lines

(vi) sends the response over the TCP connection to the requesting browser. If a user requests a file that is not present in the server, the server should return a "404 Not Found" error message.

You *may not* use external libraries that circumvent the assignment in any way such as ScaPy etc. although you can use them to verify the correctness of your code if you want.

3.1 Calling syntax

Your applications should be named `client.py` & `server.py`, and should be invoked at the command line using the following syntax:

```
python server.py
python client.py [-host] [-port] [-filename] [-timeout]
```

where the arguments are defined as follows.

- `host` (required) set it to your home IP address to run locally: host = '127.0.0.1'
- `port` (required) the port number where the connection will be established: use port 12345
- `filename` (required) the file name to query for.
- `timeout` (optional) the amount of time the client will wait for a response for before exiting. (Default: 5 seconds)

For example, to retrieve for test.txt file using the client, first enter

`python server.py` which will make the server wait for requests, then run your client

```
python client.py 127.0.0.1 12345 text.txt
```

which will result in the contents of text.txt being printed on the terminal.

3.2 Output behavior

Your program should print output to terminal using the following standard format. The first lines should summarize that a connection between the client and the server has been established and the query that has been sent. Then print out the server HTTP message and the content type of the message as well as outputting the actual file content (in the example below 'Success!!!' was the content of test.txt. Finally, do not forget to close the sockets before exiting and printing a confirmation message that the sockets have been closed. You may assume that the server, the client and the files are in the same local folder on your computer.

```
Connection: OK
Request message sent.
Server HTTP Response: HTTP/1.1 200 OK
Content-Type: text/html

Success!!!
Socket closed.
```

If no record is found then you should return a 404 error instead of printing the content of the file.

```
Connection OK.
Request message sent.
Server HTTP Response: \HTTP/1.1 404 not found
```

```
404 Not Found
Socket closed.
```

You should be able to handle text and image data. Optionally, for an additional sense of achievement try to handle mp3 and mp4 data which should really only take you an additional if statement for each type. Include a time-out for the client set at 5 seconds.

4 Important dates, deliverables, and evaluation

4.1 Demo

The demo will take place online on Sept 24th or 25th and will count for 3% of your final grade in the course. For the demo, you should be able to share your screen with the TA and execute your client and server for different queries in a zoom meeting that will happen during lab hours. You will also be asked questions about how you implemented it and basic theory related to the lab (e.g., referring to specific portions of the code), as well as what tests you performed to ensure your application behaves as required.
On non-demo weeks the TA will be available during lab time and hold virtual office hours to answer any questions related to the lab. On the first day of the lab the TA will hold a brief session explaining the lab requirements from this document and will hold a Q&A session during lab hours.

4.2 Report

The report is due at 23:59 on Sept. 28th, and it will count for 3% of your final grade. Your report should include the following items:

- The names and McGill ID numbers of both group members
- Design: Describe the design of your application. Include a requirements analysis and a diagram depicting software structure. Explain how you handle different file type requests.
- Testing: Describe how you tested your application to make sure it behaves as required
- Experiment:
    - Try to receive a file that is in the same directory as your server and client code. Screen capture the output and put it in your report. Repeat for a file that does not exist.
    - What happens after establishing a connection running your client code, but you do not close the sockets at the end? Try running multiple clients at the same time with and without closing the sockets and see what happens.
      Note: serverSocket.listen(1) you can try putting this line at the top of your server code and see what changing the parameter in the parentheses does.
    - In your client, for large messages you will not be able to download all of them in one go. You are expected to use clientSocket.recv(word_size).decode() to receive data in chunks from the server. Test how fast you can receive the data for files larger than a typical word size of e.g. 1024. You can make a big txt file and check or with a large image. Does increasing the chunk size improve the transfer speed?

To submit your report, upload a PDF file on myCourses. Only one report needs to be uploaded per group. The report should be brief – less than 4 pages. More detailed instructions will be provided next week.

4.3 Code

When submitting your report on myCourses, you should also upload a zip file containing your code. Make sure to include all source files required to compile and execute your program. Also include a readme file giving any special instructions required to run your code and mentioning what version Python you used when writing/testing your program.

5 Additional advice

Start early, the assignment is quite easy if you understand fundamental python programming, HTTP requests and how to execute from the command line.
You can use os, sys and other python libraries to make retrieving the files or displaying their content simple. The restrictions only apply to libraries that circumvent the networking part of this assignment. All networking code is to be done using Python's socket library only.